SMART User's Guide

John Y. Ngai

Department of Computer Science
California Institute of Technology
Pasadena, California 91125

Technical Report : 5118

The work described in this document was sponsored by the
Defense Advanced Research Projects Agency, ARPA order number 3771,
and monitored by the Office of Naval Research under
contract number N00014-79-C-0597

1. Introduction

SMART[1] (Simple Minded Approach Routing Tool) is a general interconnect tool for large-scale MOS custom integrated circuits.[2] The system contains knowledge about geometrical design rules. Currently, SMART supports single layer metal nMOS and CMOS/SOS processes. Multiple layer metal processes will also be supported in upcoming versions. The goal of SMART is to provide fast turn-around time wire routing.

1.1. Overview

In order to use SMART to route integrated circuit layouts, the user has to define the routing contexts in terms of *modules*. Modules are the basic structural units which SMART recognizes. A module can be as simple as a leaf cell or as complicated as an entire IC chip. Through hierarchical nesting of modules, SMART allows the user to compose large complicated circuits in terms of simpler ones.

A module declaration has four parts: attribute declarations; terminal declarations; sub-module instantiation declarations; and the netlist declarations. Attributes are information pertinent to the routing context as a whole such as the width of the power busses to be used. The remaining parts form the main "guts" of a module. Terminals are connectors located on the four sides of a module. They are the principal structures through which connections can be made to the outside. Sub-module instantiation is the mechanism to specify hierarchical nesting of modules. Netlist information defines the desired wiring connectivities to be generated by SMART.

SMART accepts a number of system commands that allow the user to modify module declarations, route modules, and produce plots. System commands will be described more fully later. First, let us look at an example.

1.2. A Simple Example

The following is an example that puts the above concepts together. All modules can be declared following this pattern.

In this example, module *sub* is a *leaf cell* and module *whole* declares a routing context consisting of two copies of *sub* placed next to each other. *Sub* could be a CIF[3] cell generated by another layout language such as Earl, or it could be a previously SMART routed module. In either case, the internal detail of *sub* is not important in the routing

---

[1] This document describes a program that is still changing. Do not assume that anything described here must be true, or if happens to be true, that it will be in the future.

[2] For a detail description of the routing algorithms used in SMART, see the author's master thesis, Caltech TR5094.

[3] Caltech Intermediate Form

context declared by *whole*. External terminals of *sub*, on the other hand, are referenced inside *whole* to declare netlist information.

```
set tech = nmos
module sub                  ; simple leaf cell
        bbox 42 49
beginm
term                        ; declare external connectors
north: n1 L NP W 2 P 7 f1 L NP W 2 P 14 n2 L NP W 2 P 28
south: s1 L NP W 2 P 7 f2 L NP W 2 P 14 s2 L NP W 2 P 28
east:  e1 L NM W 3 P 7 f3 L NM W 3 P 14 e2 L NM W 3 P 28
west:  w1 L NM W 3 P 7 f4 L NM W 3 P 14 w2 L NM W 3 P 28
endm
```

The first line

```
set tech = nmos
```

tells SMART that the circuit under description will be in nmos. The next line

```
module sub                  : a simple leaf cell
```

begins the declaration of a module named *sub*. All characters after the semi-colon and before end of line are comments. The line

```
bbox 42 49
```

defines the bounding box of the module *sub* to be 42 λ in X direction and 49 λ in Y direction. The keywords

"beginm" and "endm"

are a pair of control words to delimit the body of module declarations. The declaration of external connectors (*terminals*) starts with the keyword "term". The terminals are grouped according to the sides they belong. E.g. the line

```
north: n1 L NP W 2 P 7 f1 L NP W 2 P 14 n2 L NP W 2 P 28
```

declares three terminals : n1, f1 and n2 on the north side of *sub*. Terminals on the same side are declared in ascending X-coordinate (south, north) or Y-coordinate (west, east) values. Each terminal also admits a layer and a width specification. E.g.

```
f1 L NP W 2 P 14
```

declares f1 to be on layer NP (nmos polysilicon), of connector width 2 λ and with X-coordinate equals to 14. The Origin is fixed at the lower left corner of the module being declared. Only external terminals are declared in *sub*, since they are the only features that matter. Module *whole* can now be declared that builds up on *sub*.

```
module whole
        bbox 168 112
beginm
term
south: s0 L NP W 2 P 80
north: n0 L NP W 2 P 80
submod
sub:    { I1 21 28, I2 98 28 MX }
net
sig1:   {sub.I1.f1, sub.I2.f1}
sig2:   sub.*.f2
sig3:   sub.*.f3
sig4:   *.*.f4
so:     {s0, *.*.s1, *.*.s2}
no:     {n0, *.*.n1, *.*.n2}
we:     {*.*.w1, *.*.w2}
ea:     {*.*.e1, *.*.e2}
endm
```

Module *whole* is declared following the same structure as in *sub*. Sub-module declarations start with the keyword "submod". The line

```
sub:    { I1 21 28, I2 98 28 MX }
```

declares two instantiations of *sub* inside *whole* each with a unique *instance name* (I1 and I2). I1 is placed at with lower left corner at coordinates (21, 28). I2 is more interesting. The keyword "MX" tells SMART that *sub* should be mirrored in X direction first, and then placed with the mirrored lower left corner at coordinates (98, 28).

Net list declaration starts with the keyword "net". A signal net is simply a set of *ports* declared to be connected together by SMART. A port is a connector and is identified by three components: the name of the module where it is declared; the name of the instance where it is located and its own terminal name. E.g. sub.I1.f1 refers to the connector f1 from the instance I1 of the module *sub*. The line

```
sig1:   {sub.I1.f1, sub.I2.f1}
```

declares sig1 as a signal with two ports sub.I1.f1 and sub.I2.f1. Notice that the ports share a common terminal name. SMART allows the use of wild cards to match port names. E.g. the line

```
sig4:   *.*.f4
```

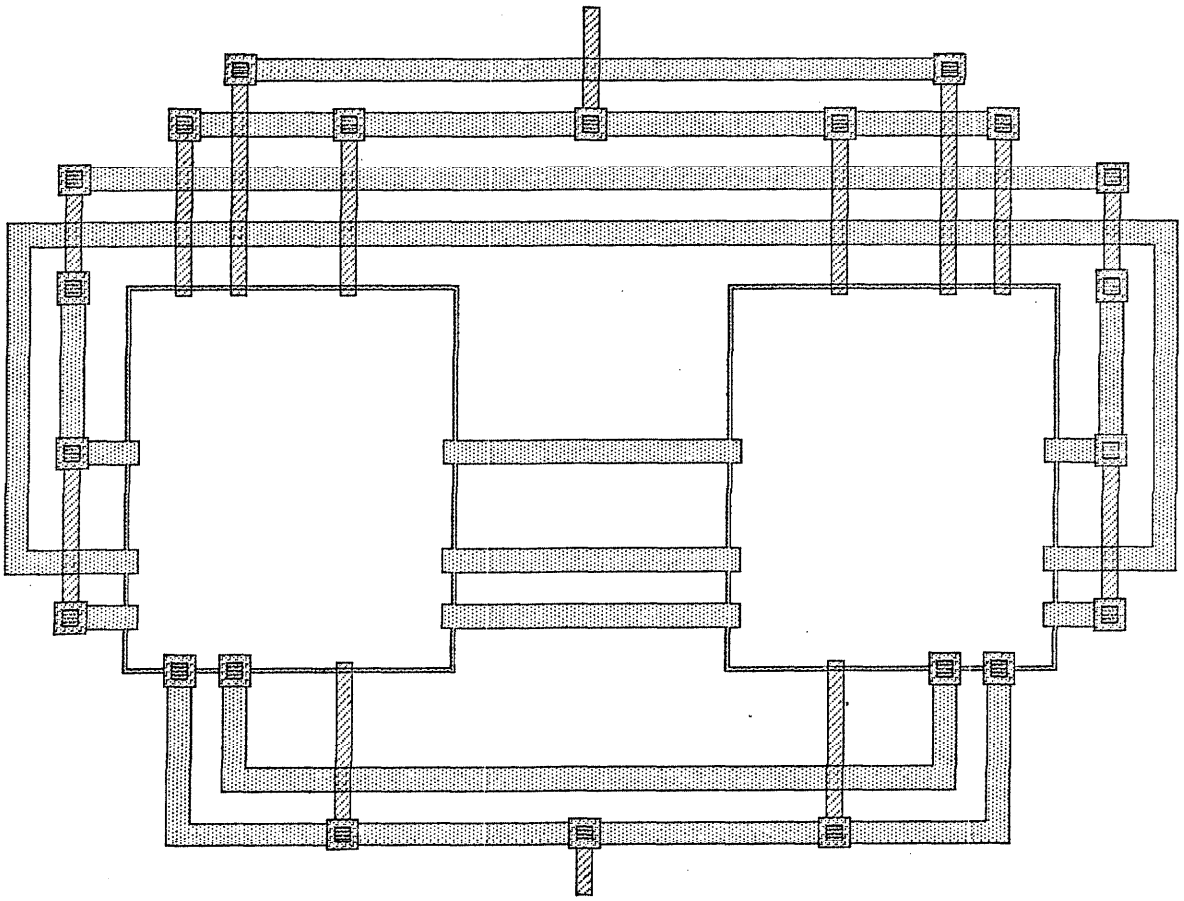

Fig 1. Module Whole.

declares sig4 as a signal with ports that have terminal name f4. Hence both sub.I1.f4 and sub.I2.f4 are to be connected as sig4. Terminals of *whole* on the other hand do not have an instance name. Reference to them can be made in two ways, e.g. the terminal s0 in *whole* can be identified as

whole.*.s0 or s0

The first form works because "*" matches even null names in SMART. The second form is simpler and should be used. With only a single name specified as a port name, SMART assumes that it is a terminal from the current module. After the modules *sub* and *whole* are declared, the command

route whole

tells SMART to route the module *whole* and produce CIF outputs. The command

plot whole

would then send the CIF produced to cifplot to get a raster plot of the module. A plot of the module *whole* produced by SMART is shown in Fig 1.

2. Model and Convention

SMART attempts to connect sets of ports declared along the four sides of the instanced modules. Modules (and their instances) are restricted to be rectangular. Terminal are connectors along sides of a module where wires generated by SMART join the internal circuitry of the module to create electrical connections. The bounding box of a module should be defined so that SMART is free to put contact cuts at the terminals if it decides to switch layers. Inserted contacts will be centered along the bounding box edges. Wires connected to terminals generated by SMART will always be perpendicular to the sides along which the terminals are declared.

SMART generates wires in Manhattan geometry. It uses a floating grid to assign tracks and columns upon which the wires lie. Two routing layers are used, one for each direction. Signal wires are generated with minimum layer widths. The grid size is determined by the geometric design rules governing the routing layers. It is selected to be large enough so that two contacts can be put side by side on adjacent grid points. In the case of single layer nMOS, it is 7 λ. The distance between two adjacent terminals of a module must also be at least as large as that of the grid size. Exact alignment of terminals with grid lines is not necessary. SMART adaptively moves the grid to match with terminals if it can, otherwise jogs will be introduced to enforce the alignments. All wires running parallel to a module's side will be at least a grid size away from the module.

With each placement of instances, SMART slices the routing plane into disjoint channels. The channels formed are maximal horizontal strips. According to the direction specified in the module attribute section, SMART will sweep across the routing plane performing switch-box routing and global signal propagation. SMART is mostly a "local" router in which global lookahead is kept minimal. Fast run time is obtained in expense of "optimal" wiring.

2.1. Routing Power

SMART supports restrictive planar routing of VDD and GND busses. In order to ensure planarity of the generated layout, SMART imposes the following restrictions :

(1) Only one polarity, either VDD or GND terminals, but not both, can be declared on any side of a module.

(2) Terminals of identical polarity CANNOT be declared on opposite sides of the same module.

Each module can declare its own power bus width to be used during the power routing phase of that module. The same width is also used in local windings of power busses in any instantiation of the module. NOTE: VDD and GND are the reserved signal names for the power and ground signals.

Power routing is carried out as a separate phase before signal routing. SMART attempts to run VDD and GND wires along opposite sides of a channel. Polarities are selected so as to minimize the number of local windings needed. A general placement will usually result in a number of narrow channels whose width is not enough to accommodate the power busses. In order to avoid routing power in these channels, an instance's bounding box can be expanded in the direction specified in the current module's attribute section.

2.2. Routing Signals

After the power routing phase, SMART attempts to route signals. SMART achieves this by routing locally in each channel in the order they are encountered as it marches across the routing plane.

In each channel, SMART attempts to achieve two goals in a greedy fashion:

(1) Perform switch-box routing of all signals declared inside the channel.

(2) Perform propagation of global signals beyond the current channel.

Routing connections are generated on a track by track basis from one edge of the channel to its opposite. Along each track SMART attempts to generate as many connections as possible. Having exhausted all its options in the current track, it marches on to the next one. Obstacles like power busses generated previously are avoided as routing proceeds.

SMART uses the tolerance ratio declared for each signal to control its global propagations. A first fit strategy is used to commit propagation locations. Once a propagation location is detected that satisfies the declared ratio, that propagation location is accepted.

If a channel is too narrow to complete all its connections, SMART will attempt to propagate the unconnected wires into the following channels if possible and try to connect there. Otherwise, errors will be reported.

3. SMART input specification

There are two types of input to SMART : module declaration and system command. Module declaration is described here and system commands will be described in section 4.

SMART allows the user to specify routing contexts hierarchically in terms of modules. Each module declaration specifies the routing to be performed inside that module. Latter modules can be declared to include former ones. All modules must be declared before their instantiations. The syntax and semantics of module declaration are summarized in the following sub-sections.

3.1. Input Syntax

In the following description, [ ] means 0 or 1; { } means 0 or more; ( ) is used for grouping; | for alternatives; - for range spanning and " " for reserved words. Upper case and lower case characters are treated separately by the system. Reserved words (between quotes), however, can be in either case. White spaces (tab, newline, space) and "," are legal separators. In practice, they are used to partition the source text for clarity. Dimensions are specified in lambda units. Coordinates are measured with origin at the lower left corner of the module. The BNF for module declaration is shown in Table 1.

3.2. Input Semantics

Each MODULE consist of:

(1) A module name.

(2) Module attributes to describe the module :

  (a) An optional direction which specifies the direction towards which the incremental routing proceeds. Default is north.

  (b) A pair of numbers (in λ) denoting the bounding box (legal area) for routing of the current module.

  (c) An optional integer for the CIF symbol number assigned to this module. With the presence of this integer, SMART treats the module as a leaf cell and ignores its internal details.

  (d) An optional number (in λ) specifying the width of the power and ground busses to be used in routing this module. If absent, the minimum layer width will be used.

(3) A BEGINM statement to denote further description of the module.

(4) An optional list of terminals. These are the external connectors of the module. The terminals are classified according to the side on which they are located. They must be specified in ascending order of their coordinates.

(5) An optional list of instantiations of formerly routed modules. Each module can be instantiated an arbitrary number of times. Each instance of these modules can then be transformed and/or expanded.

MODULEDEC	::=	MODULEATT "BEGINM" MODULEBODY "ENDM"
MODULEATT	::=	MODULENAME [SIDE] "BBOX" COORD ["SYM" INT] ["POWER" NUM]
MODULENAME	::=	"MODULE" ID
MODULEBODY	::=	[TERMINAL] [INSTANTIATION] [NETLIST]
TERMINALS	::=	"TERM" { SIDELIST }
SIDELIST	::=	SIDE ":" TERMLIST
SIDE	::=	"EAST" | "WEST" | "NORTH" | "SOUTH"
TERMLIST	::=	*TERMLIST TERM*
TERM	::=	ID [LAYER] [WIDTH] POSITION
LAYER	::=	"L" ID
WIDTH	::=	"W" NUM
POSITION	::=	"P" NUM
INSTANTIATION	::=	"SUBMOD" { INSTANCE }
INSTANCE	::=	ID ":" PLACEMENTLIST
PLACEMENTLIST	::=	"{" PLACEMENTS "}" | PLACEMENT
PLACEMENTS	::=	PLACEMENT { PLACEMENT }
PLACEMENT	::=	ID COORD { TRANSFORMTION } [EXPANSION]
TRANSFORMATION	::=	"MX" | "MY" | "ROT" INT
EXPANSION	::=	"EXP" NUM
COORD	::=	NUM NUM
NETLIST	::=	"NET" { NET }
NET	::=	ID ":" [NUM] PORTLIST
PORTLIST	::=	"{" PORTS "}" | PORT
PORTS	::=	PORT { PORT }
PORT	::=	(ID|*) | (ID|*) "." (ID|*) | (ID|*) "." (ID|*) "." (ID|*)
ID	::=	(a-zA-Z) { (a-zA-Z0-9) }
INT	::=	(0-9) { (0-9) }
NUM	::=	*(0-9) { (0-9) } [.] { (0-9) }*

Comments can be inserted by preceding the text of the comment with a semi-colon (";"). The remainder of the line which contains ";" will be ignored by the parser.

Table 1. The BNF for module declaration.

(6) An optional list of nets to describe the connections of ports within the module. These ports are either declared in the current module declaration or are declared by previous declarations and their current instantiations.

(7) An ENDM statement to denote end of module declaration.

To represent the lowest level leaf cell, simply ignore the instantiation and net list declarations. A module with its CIF symbol declared in the module attribute section denotes the existence of a CIF symbol (of that number) that corresponds exactly to the internal details of that module and suppresses its routing. Absence of a CIF symbol number will prompt SMART to attempt to route the module and in the case of leaf cells, SMART will create a phantom CIF cell of appropriate size. This is useful during routing to suppress unnecessary detail.

Each TERMINAL consists of:

(1) A terminal name.

(2) An optional specification of layer and width for that terminal. If absent, SMART will use the topmost layer or the last declared layer. The width will then be the minimum layer width.

(3) A number (in λ) specifying the location of the pin with respect to its side.

Each INSTANTIATION consists of:

(1) The name of the module being instanced.

(2) An instance name for this instance.

(3) Two numbers denoting the coordinates of the lower left corner of the instance to be placed.

(4) An optional list of transformations and/or expansion. Transformations are performed before placement of the instance.

Each TRANSFORMATION consists of:

Mirroring in either X (MX) or Y (MY) direction, or rotations of 90 degrees (ROT x). Clockwise rotations are specified by negative integers (e.g ROT -2 means clockwise 180 degrees) and anticlockwise rotations by positive integers. The transformations are performed in the order as they are specified.

EXPANSION consists of

An integer (in λ) denoting how much the BBOX of the instance is to be expanded. Expansion is in the same direction as that specified in the module attribute section. It is used during routing of power busses to avoid narrow channels.

Each NET consists of:

(1) A net name. This is used in reporting errors and other performance statistics.

(2) An optional number specifying the global routing tolerance ratio for this net. A tolerance ratio of 1.0 is "optimal" and >1.0 is "sub-optimal". Default is 1.0.

(3) A list of ports. In general a port is uniquely defined by three components: the name of a module being instanced; the name of a particular instance; the name of a terminal. It is represented in the form modName.instName.termName. Ports declared external to the current module can however specified uniquely by its terminal name. In the case where an instanced module is instanced only once in the current module, the instance name can be omitted. Wild card specifications is used to denote all possible matches which can replace the "*"s. E.g. If all terminals intended to be connected as the signal "phi1" are named "phi1", "*.*.phi1" will match them all.

4. Using SMART

SMART is written in C and runs under Berkeley UNIX Version 4.2.[4] It can be invoked with an optional list of input file arguments. These files are read in one by one in the same order as they appear in the argument list. SMART will then prompt the user for further commands.

Most of the system commands to SMART accept an argument string. The argument string can be either an identifier or a string. A string is simply a sequence of characters enclosed by quots ("). Strings are used when the argument contains spaces or other strange characters. Commands which allow the argument string to be either a module name or a file name will always treat it as a module name first. Hence it is better to use different names for files and their modules.

The system commands are summarized below:

READ *fileName*

Read in the named file. Perform syntax and semantics check on modules declared in it. Execute system commands contained in it.

ROUTE [*-spdf*] *moduleName*

Recursively route the named module and its sub-modules. Previously routed sub-modules are simply instantiated, otherwise, they are routed first and then instantiated. Phantom CIF cells are created if sub-module routing is unsuccessful. This allows routing of top-level module to continue without interruption. Routing is suppressed if syntax or semantic errors are detected in the module declarations.

[4] UNIX is a trademark of Bell Laboratories.

The optional flags are :

-d Produce CIF symbols for the channels sliced by SMART on the Geometric Comment Layer. This is useful in checking module placements.

-p Enable routing of power and ground busses. This is useful in further checking of module placements. Narrow channels can usually be discovered and appropriate expansions inserted.

-s Enable routing of signals.

-f Force routing of all sub-modules regardless of whether they were routed already or not.

If no flag is specified, the default is "-ps".

SAVE

moduleName

Recursively save the CIF outputs for the module and all its sub-modules. The CIF output file name is the same as the module name with ".cif" extension.

[*fileName*]

All CIF output produced so far up to this command are saved into the named file. If no file name is given, the name stored in the system variable "CIF" is used as the file name.

EDIT

moduleName

The editor declared in the system variable "EDITOR" is invoked with the section of source text defining the named module. Upon exiting the editor, the edited source text is read in again and replaces its previous declaration. The source file containing the named module is also updated.

fileName

Here the editor is invoked with the entire named source file. Upon exiting the editor, the edited source file is read in again and modules declared in the source file replace their previous declarations.

CIFP

moduleName

Pass execution to CIFP with the CIF outputs of the named module and its submodules. Returns to SMART upon exiting CIFP. The name of the output device used in CIFP is stored in the system variable "DEV".

fileName

Same as above but with the named file as argument to CIFP.

PLOT

moduleName

SMART forks up a child process. Child pipes to CIFPLOT the CIF outputs of the named module and its sub-modules. Parent returns. The flags argument sent to CIFPLOT is stored in the system variable "ARG".

fileName

Same as above but with the named CIF file as argument to CIFPLOT.

LIST

moduleName

Type out detail information of the named module. The information typed is retrieved from the internal data structures in SMART. This is useful in comparing what the user thinks and what SMART thinks regarding the module.

fileName

All modules declared in named file is listed together with their status and CIF symbol numbers.

sysVarName

The value of the named system variable is listed. There are six system variables :

DEV – Output device used by CIFP. Default is "/dev/gigi".

ARG – Flags argument sent to CIFPLOT. Default is "-ltext,bbox".

CIF – CIF output file name. Default is "smart.cif".

TECH – Technology of current context. Default is "nmos".

EDITOR – Editor invoked during an edit command. Default is "ked".

LAMBDA -- Lambda size of current context in microns. Current default is "2.0".

The system variable names can be specified in either lower case or upper case.

SET *sysVarName = value*

The named system variable is set to the specified value.

! *shellCommand*

Pass to UNIX Shell the command string for execution.

CLEAR

Flush SMART's work space. All previous module declarations are erased.

?/HELP

Type a brief summary of system commands available.

<control-D>

Save CIF outputs into file specified in "CIF" if they have not been saved already. Perform clean-up and quit.

The default values for the system variables can be changed by defining the corresponding environment variables. When SMART is invoked, it will scan the environment variable list and replace the corresponding default values. Since UNIX Shell is case sensitive, only upper case environment variables are recognized by SMART.

To obtain a reasonably compact layout, the user can start with a loose placement of the instances. Route it, and then tighten up the placements according to the routing layout SMART produced by using EDIT. Several iterations may be needed to arrive at a "compact" layout.

5. Another Example

The following is a more complicated example of a module with three sub-modules instanced. A plot of the module produced by SMART is shown in Fig 2.

```
module mod1
        power  4
        bbox   77 175
beginm
term
north:  f1 L ND P 14 n2 P 28 n3 P 42 n1 P 49 vdd L NM W 4 P 70
south:  s1 L NP P 14 f2 P 28 s2 P 35 s3 P 42 gnd L NM W 6 P 70
east:   e1 L NM P 14 f3 P 35 vdd W 4 P 70 r1 L ND P 91
        r2 L NP P 98 r3 L ND P 105 e2 L NM P 154 e3 P 161
west:   w1 L NM P 28 f4 P 63 w2 P 91 w3 P 126
endm;

module mod2
        power  4
        bbox   77 63
beginm
term
north:  r1 L NP P 7 r2 P 21 n1 L ND P 49 vdd L NM W 6 P 70
south:  s1 L ND P 7 f2 P 14 s2 P 28 gnd L NM W 6 P 70
east:   e1 L NM P 7 f3 P 14 e2 P 28 e3 P 42
west:   w1 L NM P 7 f4 P 14 w2 P 28 w3 P 56
endm
```

```
module mod3
        power  4
        bbox   77 63
beginm
term
north:  f1 L ND P 14 n2 P 28 n1 P 56 vdd L NM W 6 P 70
south:  r3 L NP P 14 s2 L ND P 35 gnd L NM W 6 P 70
east:   e1 L NM P 7 f3 P 14 e2 P 28 t4 P 35 e3 P 49
west:   w1 L NM P 7 f4 P 14 w2 P 28 w3 P 49
endm

module complex
        north
        power  12
        bbox   313 315
beginm
term
north:  n1 L NM P 14 f1 P 63 n2 L ND P 112 n3 P 172
south:  s1 L ND P 28 s2 P 70 f2 P 98 s3 P 126
west:   gnd L NM W 12 P 20
east:   t2 L NM P 91 t3 L NP P 175 vdd L NM W 12 P 240;
submod
mod1:   I1 21 56 EXP 21              ; to avoid narrow channel
mod3:   I1 170 189
mod2:   I1 170 56
net
vdd:    *.*.vdd
gnd:    *.*.gnd
; all the rest signals have tolerance ratio 1.0!
sig1:   *.*.s1
sig2:   *.*.s2
sig3:   *.*.f2
sig4:   *.*.s3
sig5:   *.*.n1
sig6:   *.*.n2
sig7:   *.*.f1
sig8:   *.*.n3
sig9:   { mod1.e1, mod2.w1, mod3.w1 }
```

```
sig10: { mod1.e2, mod2.w2, mod3.w2 }
sig11: { mod1.e3, mod2.w3, mod3.w3 }
sig12: { mod1.f3, mod2.f4, mod3.f4 }
sig13: { mod2.e1, mod3.e1 }
sig14: { mod2.e2, mod3.e2 }
sig15: { mod2.e3, mod3.e3 }
sig16: { mod2.f3, mod3.f3 }
sig17: { mod1.w1, mod1.w2 }
sig18: { mod1.f4, mod1.w3 }
sig19: { *.*.t2, *.*.t3, *.*.t4 }
sig20: *.*.r1
sig21: *.*.r2
sig22: *.*.r3
endm
```

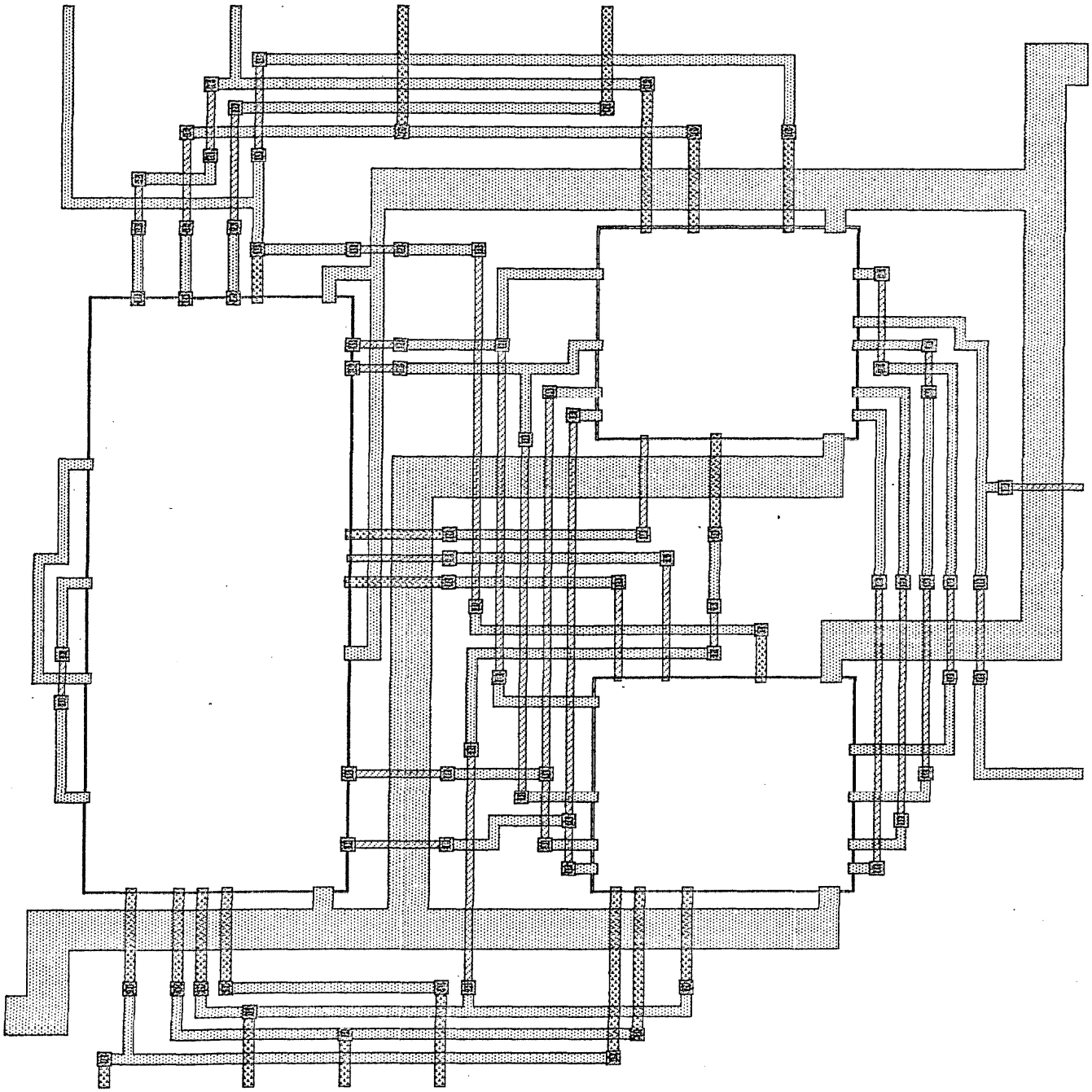

Fig 2. Module Complex.